Лабораторная работа 10.

Черновик 0.55

Цель работы – приобрести навыки работы с «универсальным» (информационная часть представлена указателем на void) линейным односвязным списком.

Студенты должны получить и закрепить на практике следующие знания и умения:

1. Работа с динамической памятью.
2. Работа с бестиповыми указателями и указателями на функцию.
3. Обработка линейного односвязного списка (добавление элемента, удаление элемента, поиск элемента, комбинированные действия).
4. Обработка текстовых файлов.
5. Организация корректной работы с ресурсами (динамически выделенная память, файловые описатели).
6. Использование в программе аргументов командной строки.
7. Контроль правильности работы с динамической памятью с помощью специального ПО.

1. Общее задание

1. Исходный код лабораторной работы располагается в ветке lab_10. В этой ветке создается папка lab_10, в которой располагается исходный код программа.
2. Исходный код должен соответствовать правилам оформления исходного кода.
3. Решение каждой задачи оформляется как многофайловый проект.
4. Для проверки правильности решения задачи реализуются как модульный, так и функциональные тесты. Входные данные должны располагаться в файлах in_z.txt (возможно таких файлов будет несколько), выходные out_z.txt, где z – номер тестового случая. Тестовые данные готовятся и помещаются под версионный контроль еще до того, как появится реализация задачи.
5. Общие требования к реализации задачи:
   - Информационная часть элемента списка представлена указателем на void.

```
typedef struct node node_t;

struct node
{
    void *data;
    node_t *next;
};
```

   - Список формируется на основе данных, которые читаются из текстового файла. В качестве данных может, например, выступать информация о студенте: фамилия и год рождения. Область данных выбирается самостоятельно и отличается от указанного примера.
   - Элементы списка данными "не владеют", т.е. хранят лишь указатель на них. При удалении элемента из списка данные не удаляются.
   - Для решения некоторых задач требуется функция сравнения элементов. Сравнение элементов реализуется как отдельная функция. Функции, нуждающиеся в сравнении элементов, получают в качестве параметра указатель на функцию сравнения.
   - Имена файлов, выполняемая операция и т.п. указываются через параметры командной строки, результаты работы записываются в файл.

6. В качестве одного из необходимых результатов работы должно быть приведено условие осмысленной задачи в выбранной предметной области, которая может быть решена с использованием разработанных вами функций (см. индивидуальное задание) и для решения которой достаточно того объема данных, которые считываются из исходного файла. Также должна быть написана программа, решающая поставленную задачу.

2. Индивидуальное задание

Индивидуальные задачи выбираются студентом самостоятельно. При желании преподаватель, проводящий лабораторные работы, может выдавать их по вариантам. Условная сложность каждой задачи указанна в скобках после условия задачи.

В случае обнаружения заимствований по усмотрению преподавателя может быть выдано другое задание или выставлено 0 баллов без права повторной сдачи этой лабораторной в течение семестра.

Задачи на работу с одним элементом списка

Необходимо решить любые две задачи.

- Напишите функцию поиска элемента в списке (нужна функция сравнения элементов). (1)

```
node_t* find(node_t *head, const void *data,
                  int (*comparator)(const void*, const void *));
```

- Напишите функцию pop_front, которая возвращает указатель на данные из элемента, который расположен в "голове" списка. При этом из списка сам элемент удаляется. (1)

```
void* pop_front(node_t **head);
```

- Напишите функцию pop_end, которая возвращает указатель на данные из элемента, который расположен в "хвосте" списка. При этом из списка сам элемент удаляется. (1)

```
void* pop_back(node_t **head);
```

- Напишите функцию insert, которая вставляет элемент перед указанным элементом списка (в качестве параметров указываются адреса обоих элементов). (2)

```
void insert(node_t **head, node_t *elem, node_t *before);
```

Задачи на работу с целым списком

Необходимо решить одну любую задачу.

- Напишите функцию copy, которая по указанному списку создает его копию (данные при этом не копируются). (1)

```
int copy(node_t *head, node_t **new_head);
// функция возвращает код ошибки, потому что она выделяет память
```

- Напишите функцию append, которая получает два списка a и b, добавляет список b в конец a. Список b при этом оказывается пустым. (1)

```
void append(node_t **head_a, node_t **head_b);
```

- Напишите функцию remove_duplicates, которая получает упорядоченный список и оставляет в нем лишь первые вхождения каждого элемента. Совпадение определяется с помощью функции сравнения элементов. При удалении элемента списка данные не удаляются. (2)

```
void remove_duplicates(node_t **head,
                       int (*comparator)(const void*, const void*));
```

- Напишите функцию reverse, которая обращает список. Идеи реализации:
  - Использование pop_front и двух списков. (1)
  - Использование 3-х указателей на соседние элементы списка. (2)
  - Рекурсия. (2)

```
node_t* reverse(node_t *head);
// возвращается «новая» голова
```

Сортировка списка

```
node_t* sort(node_t *head, int (*comparator)(const void *, const void *));
// возвращается «новая» голова
```

Необходимо реализовать одну из двух сортировок.

Сортировка вставками

Напишите функцию sorted_insert, которая получает упорядоченный список, и элемент, который нужно вставить в этот список, чтобы не нарушить его упорядоченности.

```
void sorted_insert(node_t **head, node_t *element,
                   int (*comparator)(const void *, const void *));
```

Напишите функцию sort, которая получает список и упорядочивает его по возрастанию, используя функцию sorted_insert. (3)

Сортировка слиянием

Напишите функцию front_back_split, которая получает список и делит его на две половины. Если в списке нечетное число элементов, "серединный" элемент должен попасть в первую половину.

```
void front_back_split(node_t* head, node_t** back);
```

Напишите функцию sorted_merge, которая получает два упорядоченных списка и объединяет их в один.

```
node_t* sorted_merge(node_t **head_a, node_t **head_b,
                       int (*comparator)(const void *, const void *));
```

```
// Списки становятся пустыми, элементы из них «переходят» в упорядоченный
```

Используя функции front_back_split и sorted_merge напишите функцию sort, которая реализует рекурсивный алгоритм сортировки слиянием. (4)